

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

X.785

(07/2021)

SERIES X: DATA NETWORKS, OPEN SYSTEM
COMMUNICATIONS AND SECURITY

OSI management – Management functions and ODMA
functions

Guidelines for defining REST-based managed objects and management interfaces

Recommendation ITU-T X.785

ITU-T X-SERIES RECOMMENDATIONS
DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY

PUBLIC DATA NETWORKS	
Services and facilities	X.1–X.19
Interfaces	X.20–X.49
Transmission, signalling and switching	X.50–X.89
Network aspects	X.90–X.149
Maintenance	X.150–X.179
Administrative arrangements	X.180–X.199
OPEN SYSTEMS INTERCONNECTION	
Model and notation	X.200–X.209
Service definitions	X.210–X.219
Connection-mode protocol specifications	X.220–X.229
Connectionless-mode protocol specifications	X.230–X.239
PICS proformas	X.240–X.259
Protocol Identification	X.260–X.269
Security Protocols	X.270–X.279
Layer Managed Objects	X.280–X.289
Conformance testing	X.290–X.299
INTERWORKING BETWEEN NETWORKS	
General	X.300–X.349
Satellite data transmission systems	X.350–X.369
IP-based networks	X.370–X.379
MESSAGE HANDLING SYSTEMS	X.400–X.499
DIRECTORY	X.500–X.599
OSI NETWORKING AND SYSTEM ASPECTS	
Networking	X.600–X.629
Efficiency	X.630–X.639
Quality of service	X.640–X.649
Naming, Addressing and Registration	X.650–X.679
Abstract Syntax Notation One (ASN.1)	X.680–X.699
OSI MANAGEMENT	
Systems management framework and architecture	X.700–X.709
Management communication service and protocol	X.710–X.719
Structure of management information	X.720–X.729
Management functions and ODMA functions	X.730–X.799
SECURITY	X.800–X.849
OSI APPLICATIONS	
Commitment, concurrency and recovery	X.850–X.859
Transaction processing	X.860–X.879
Remote operations	X.880–X.889
Generic applications of ASN.1	X.890–X.899
OPEN DISTRIBUTED PROCESSING	X.900–X.999
INFORMATION AND NETWORK SECURITY	X.1000–X.1099
SECURE APPLICATIONS AND SERVICES (1)	X.1100–X.1199
CYBERSPACE SECURITY	X.1200–X.1299
SECURE APPLICATIONS AND SERVICES (2)	X.1300–X.1499
CYBERSECURITY INFORMATION EXCHANGE	X.1500–X.1599
CLOUD COMPUTING SECURITY	X.1600–X.1699
QUANTUM COMMUNICATION	X.1700–X.1729
DATA SECURITY	X.1750–X.1799
IMT-T SECURITY	X.1800–X.1819

For further details, please refer to the list of ITU-T Recommendations.

Recommendation ITU-T X.785

Guidelines for defining REST-based managed objects and management interfaces

Summary

Recommendation ITU-T X.785 defines a set of guidelines for managed object modelling and a management interface for representational state transfer (REST)-based network management. It is part of a framework for REST-based network management interfaces. It specifies how REST-based management interfaces should be defined. It covers the generic accessing methods of REST-based managed objects, accessing methods for specific managed object (Mos), information modelling in REST / hypertext transfer protocol (HTTP) and YAML ain't markup language (YAML) / JavaScript object notation (JSON) schemas. Some HTTP requests/responses and YAML/JSON schemas are provided for defining some basic data types: generic managed object (MO) and generic MO accessing methods.

History

Edition	Recommendation	Approval	Study Group	Unique ID*
1.0	ITU-T X.785	2021-07-29	2	11.1002/1000/14745

Keywords

REST, framework, guidelines, network management.

* To access the Recommendation, type the URL <http://handle.itu.int/> in the address field of your web browser, followed by the Recommendation's unique ID. For example, <http://handle.itu.int/11.1002/1000/11830-en>.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure, e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents/software copyrights, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the appropriate ITU-T databases available via the ITU-T website at <http://www.itu.int/ITU-T/ipr/>.

© ITU 2021

All rights reserved. No part of this publication may be reproduced, by any means whatsoever, without the prior written permission of ITU.

Table of Contents

		Page
1	Scope	1
2	References.....	1
3	Definitions	2
3.1	Terms defined elsewhere	2
3.2	Terms defined in this Recommendation.....	2
4	Abbreviations and acronyms	2
5	Conventions	3
6	Overview of a REST-based management framework	4
6.1	Overview	4
6.2	Resources.....	4
6.3	Definition languages of REST-based interface	5
7	Principles for REST-based interface design.....	5
8	Definition of a generic managed object using YAML schema	6
8.1	REST role in management interfaces	6
8.2	Definition of managed objects using JSON/YAML schema	7
9	Accessing methods for managed objects.....	12
9.1	Generic MO accessing methods	12
9.2	Design guidelines for specific MO class accessing methods	13
10	Information modelling guidelines for REST-based interfaces	15
10.1	Resource Modeling.....	15
10.2	Attribute.....	16
10.3	Name conventions for MOCs, attributes and data types	16
10.4	Other guidelines.....	16
11	Compliance and conformance	16
11.1	Standards document compliance	16
11.2	System conformance	17
11.3	Conformance statement guidelines.....	17
Annex A	Common REST-based YAML/JSON schema definitions	18
A.1	YAML schema definitions for the generic managed object and common data types	18
A.2	YAML/JSON schema definitions for common object accessing methods	21
Appendix I	An example of REST-based interface definitions for resource	38
I.1	An example showing the CRUD definitions for a specific resource.....	38
Appendix II	Usage examples of the ContainmentRelationshipType and AssociationRelationType.....	44
Appendix III	Background for REST and HTTP technologies.....	46
III.1	Background.....	46
III.2	Short review of REST and HTTP	46

	Page
III.3 Benefits of introducing REST into network management domain	48
Bibliography.....	49

Recommendation ITU-T X.785

Guidelines for defining REST-based managed objects and management interfaces

1 Scope

The network management architecture defined in [ITU-T M.3010] introduces the use of multiple management protocols. So far, the guidelines for the definition of managed objects (GDMO) / common management information protocol (CMIP), common object request broker architecture (CORBA) / Internet inter-ORB protocol (IIOP), structure of management information (SMI) / simple network management protocol (SNMP), web services / simple object access protocol (SOAP) are possible choices at the application layer. Based on the management interface specification methodology defined in [ITU-T M.3020], more technology-based paradigms can be introduced into network management interfaces, and REST/HTTP is now an additional paradigm for network management.

This Recommendation sets out a framework for defining how interfaces supported by management systems and network elements should be modelled using JSON/YAML schemas. It is within the scope of this Recommendation to provide the following guidelines or instructions:

- principles for REST interface designs;
- containment, association and inheritance relationship and naming rules for managed entities;
- generic accessing methods for managed objects;
- guidelines for defining accessing methods for specific resources;
- information modelling guidelines for REST-based interfaces;
- common data type definitions.

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

- [ITU-T M.3010] Recommendation ITU-T M.3010 (2000), *Principles for a telecommunications management network*.
- [ITU-T M.3020] Recommendation ITU-T M.3020 (2017), *Management interface specification methodology*.
- [ITU-T M.3160] Recommendation ITU-T M.3160 (2008), *Generic, protocol-neutral management information model*.
- [ITU-T M.3701] Recommendation ITU-T M.3701 (2010), *Common management services – State management – Protocol neutral requirements and analysis*.
- [ITU-T X.701] Recommendation ITU-T X.701 (1997), *Information technology – Open Systems Interconnection – Systems management overview*.
- [ITU-T X.703] Recommendation ITU-T X.703 (1997), *Information technology – Open Distributed Management Architecture*.

[RFC 3986]	IETF RFC 3986 (2005), <i>Uniform Resource Identifier (URI): Generic Syntax</i> .
[RFC 5789]	IETF RFC 5789 (2010), <i>PATCH Method for HTTP</i> .
[RFC 6585]	IETF RFC 6585 (2012), <i>Additional HTTP Status Codes</i> .
[RFC 6901]	IETF RFC 6901 (2013), <i>JavaScript Object Notation (JSON) Pointer</i> .
[RFC 6902]	IETF RFC 6902 (2013), <i>JavaScript Object Notation (JSON) Patch</i> .
[RFC 7230]	IETF RFC 7230 (2014), <i>Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing</i> .
[RFC 7231]	IETF RFC 7231 (2014), <i>Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content</i> .
[RFC 7232]	IETF RFC 7232 (2014), <i>Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests</i> .
[RFC 7396]	IETF RFC 7396 (2014), <i>JSON Merge Patch</i> .

3 Definitions

3.1 Terms defined elsewhere

This Recommendation uses the following terms defined elsewhere:

3.1.1 agent [ITU-T M.3020]: Encapsulates a well-defined subset of management functionality. It interacts with managers using a management interface. From the manager's perspective, the agent behaviour is only visible via the management interface.

3.1.2 managed object class [ITU-T X.701]: A named set of managed objects sharing the same (named) sets of attributes, notifications, management operations (packages), and which share the same conditions for presence of those packages.

3.1.3 manager [ITU-T M.3020]: Models a user of agent(s) and it interacts directly with the agent(s) using management interfaces. Since the manager represents an agent user, it gives a clear picture of what the agent is supposed to do. From the agent perspective, the manager behaviour is only visible via the management interface.

3.1.4 notification [ITU-T X.703]: An interaction for which the contract between the invoking object (client) and the receiving object (server) is restricted to the ability of the server to receive the contents of information sent by the client.

3.2 Terms defined in this Recommendation

This Recommendation does not define any new terms.

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

API	Application Programming Interface
BNF	Backus-Naur Form
CMIP	Common Management Information Protocol
CORBA	Common Object Request Broker Architecture
CRUD	Create, Retrieve, Update, Delete
DN	Distinguished Name

GDMO	Guidelines for the Definition of Managed Objects
HTTP	Hypertext Transfer Protocol
IOP	Internet Inter-ORB Protocol
IT	Information Technology
JSON	JavaScript Object Notation
MO	Managed Object
MOC	Managed Object Class
MOI	Managed Object Instance
OAS	OpenAPI Specification
OSI	Open System Interconnection
RDN	Relative Distinguished Name
REST	Representational State Transfer
RPC	Remote Procedure Call
SMI	Structure of Management Information
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
TMN	Telecommunications Management Network
URI	Unified Resource Identifier
XML	extensible Markup Language
YAML	YAML Ain't Markup Language

5 Conventions

A few conventions are followed in this Recommendation to make the reader aware of the purpose of the text. While most of the Recommendation is normative, paragraphs succinctly stating mandatory requirements to be met by a management system (managing and/or managed) are preceded by a boldface "R" enclosed in parentheses, followed by a short name indicating the subject of the requirement and a number. For example:

(R) EXAMPLE-1 An example mandatory requirement.

Requirements that may be optionally implemented by a management system are preceded by an "O" instead of an "R". For example:

(O) EXAMPLE-2 An example optional requirement.

The requirement statements are used to create compliance and conformance profiles.

Examples of JSON and YAML schemas are included in this Recommendation and normative JSON or YAML schema specifying the data types, base classes and other modelling constructs of the framework are included in Annex A. The JSON/YAML schemas are written in a 10 point courier typeface:

A JSON schema example	<pre>{ "title": "root", "items": { "title": "array item" } }</pre>
A YAML schema example	<pre>SomeType: type: object properties: attr1: type: string attr2: type: string enum: - e1 - e2</pre>

6 Overview of a REST-based management framework

6.1 Overview

REST-based technologies have been widely used in the information technology (IT) industry. Appendix III provides more information on the features of REST technology. REST technology is similar to web services technology and can be used in network management interfaces. The REST technology uses a resource-oriented approach to define information entities, and unified resource identifiers (URIs) ([RFC 3986]) for entity identification, and the corresponding operations are also defined with a close relation with the resource URIs.

This Recommendation sets up a framework for defining how interfaces supported by management systems and network elements should be modelled using REST application programming interface (API) and JSON/YAML schemas (see [RFC 6901], [RFC 6902], and [b-OAI-OAS3]).

The complete REST-based management framework includes the following aspects:

- 1) Managed objects and interface definition guidelines:
 - definition of managed object classes using YAML schema;
 - inheritance, containment and association relationships of managed objects (Mos);
 - accessing methods for managed object instances (MOIs);
 - information modelling guidelines for REST-based interfaces;
- 2) REST-based supporting services for network management:
 - definition of a REST-based notification service;
 - definition of a REST-based heartbeat service;
 - definition of a REST-based containment service.

This Recommendation mainly deals with managed objects and interface definition guidelines. REST-based supporting services will be dealt with in other Recommendations.

6.2 Resources

Hypertext transfer protocol (HTTP) ([RFC 7230]) uses a different terminology based on the notion of resources, as defined in clause 2 of [RFC 7231]. Each resource is represented by a resource representation as defined in clause 3 of [RFC 7231]. Valid resource representations are e.g. extensible markup language (XML) instance documents or JSON instance documents.

Resources can be classified according to their structure and behaviour into resource archetypes. This helps to specify clear and understandable interfaces. The following three archetypes are defined (also aligned with [b-3GPP TS 32.158]):

- **Document resource:** This is the standard resource containing data in form of name-value pairs and links to related resources. This kind of resource typically represents a real-world object or a logical concept.
- **Collection resource:** A collection resource is grouping resources of the same kind. The resources below the collection resource are called items of the collection. An item of a collection is normally a document resource. Collection resources typically contain links to the items of the collection and information about the collection like the total number of items in the collection. Collection resources can be further distinguished into server-managed and client-managed resources. Collection resources are also known as container resources.
- **Operation resource:** Operation resources represent executable functions. They may have input and output parameters. Operation resources allow some kind of fallback to a remote procedure call (RPC) style design in case application specific actions cannot be mapped easily to create, retrieve, update, delete (CRUD) style operations.

6.3 Definition languages of REST-based interface

This Recommendation follows the OpenAPI specification (OAS, see [b-OAI-OAS3]) to define REST-based interfaces. The OpenAPI specification (OAS) is developed by the OpenAPI initiative, which defines a standard, programming language-agnostic interface description for REST APIs, which allows both humans and computers to discover and understand the capabilities of a service without requiring access to a source code, additional documentation, or inspection of network traffic. There are two languages that can be used in OAS: JSON and YAML, and the relationship between them is described below.

YAML is a human-friendly, cross language, unicode based data serialization language designed around the common native data types of agile programming languages. It is broadly useful for programming needs ranging from configuration files to internet messaging to object persistence to data auditing.

Both JSON and YAML aim to be human readable data interchange formats. However, JSON and YAML have different priorities. JSON's foremost design goal is simplicity and universality. Thus, JSON is trivial to generate and parse, at the cost of reduced human readability. It also uses a lowest common denominator information model, ensuring any JSON data can be easily processed by every modern programming environment.

In contrast, YAML's foremost design goals are human readability and support for serializing arbitrary native data structures. Thus, YAML allows extremely readable files but is more complex to generate and parse. In addition, YAML ventures beyond the lowest common denominator data types, requiring more complex processing when crossing between different programming environments.

YAML can therefore be viewed as a natural superset of JSON, offering improved human readability and a more complete information model. This is also the case in practice; every JSON file is also a valid YAML file. This makes it easy to migrate from JSON to YAML if/when the additional features are required.

Within this Recommendation, both JSON and YAML schemas will be used for interface definitions.

7 Principles for REST-based interface design

This clause identifies some interface design considerations that should be addressed by this framework through REST interfaces. It provides the modelling principles for REST-based managed objects (MO) and their accessing methods.

The REST-based design considerations related to REST APIs and JSON/YAML schemas and modelling concerns super-classes, naming of managed objects and resource-oriented interfaces, operations and notifications.

This Recommendation defines a lightweight generic use of REST-based interface design patterns. The management and controlling functions are defined using HTTP methods, not just for an individual management object class.

The framework has the following principles to define a REST-based management information model and interfaces.

- All interface interactions are defined as HTTP methods, each operation includes a request and an optional corresponding response when needed.
- Each managed object class (MOC) is defined as a resource when exchanged through the management interface, and each attribute or state of the MOC is defined as a property in the resource.
- The naming of MOC instances follows the concept of a URI, which can be accessed using an HTTP request.
- There are four basic accessing methods for managed objects in the traditional telecommunications management network (TMN) management paradigm, which are: createMO, deleteMO, getMOAttributes, and setMOAttribute. These methods are redefined in this management framework using HTTP POST, HTTP DELETE, HTTP GET, and HTTP PUT/PATCH. These methods are applicable for every MOC instances, and the URI is used to indicate which instances are accessed using these methods.
- Other interface control functions are defined as HTTP POST methods against a specific resource.
- Common data types are defined in JSON/YAML schemas which can be shared by application-specific interface definitions.

8 Definition of a generic managed object using YAML schema

8.1 REST role in management interfaces

To support the software objects representing manageable resources, a base class is defined for use in modelling network resources. Other MOCs (managed object class) in information models must be derived from this base class in order to operate within this framework. Some generic accessing methods and some other extended functions are defined to provide interfaces to manage MOs.

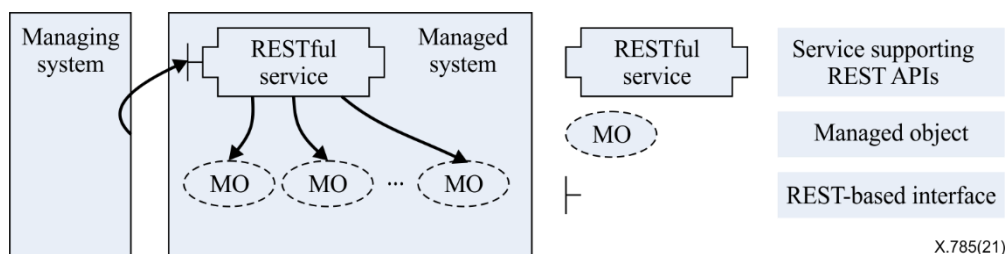


Figure 1 – RESTful services role

Figure 1 shows how a managing system accesses a managed system that supports a RESTful services interface. A RESTful services interface acts as an intermediate entity that enables a managing system to manage proper MOs in a managed system representing manageable resources.

8.2 Definition of managed objects using JSON/YAML schema

An MO is an open system interconnection (OSI) management view of a resource that is subject to management, such as a connection or an item of physical equipment. It is the abstraction of such a resource that represents its properties for the purpose of management. An MO may also include attributes that provide information used to characterize itself and operations that represent its behaviours. The purpose of the framework is to provide a collection of capabilities to manage these MOs. MOs need some approaches to describe their properties and behaviours. In REST-based technology, an MO is a managed entity that represents a manageable resource in terms of shared state and behaviour where state and behaviour are separated through outsourcing of the behaviour to an assigned so-called "managing entity" (e.g., a service and its interface) that takes a steward role with regard to the behaviours of its allocated managed entities. Since an MO's state and behaviour can be separated, the state can be described by JSON/YAML schema and behaviour by REST APIs. One important benefit of using a JSON/YAML document to store an MO's state is that REST APIs can also use JSON/YAML schema to describe the data type of its exchanged messages, and these JSON/YAML based MOs' information can be exchanged without any modification.

8.2.1 Definition of a generic managed object class

A managed object class is a further abstraction of managed objects. All network resources have some common attributes and all MOCs shall inherit, either directly or indirectly, from a superclass, namely a ManagedObject_C (the suffix "_C" indicates it represents an MOC, not just a data type). Using ManagedObject_C to define new MOCs will be easier and faster and provide better maintenance. As mentioned above, all MOCs are described in JSON/YAML schema and the data type of ManagedObject_C is given in Table 1 and the attributes can be found in Table 2.

Table 1 – Data type of superclass ManagedObject_C

<pre>ManagedObject_C: type: object required: - objectClass - objectInstance properties: objectClass: type: string objectInstance: type: string format: uri creationSource: \$ref: '#/components/schemas/SourceIndicatorType'</pre>
<pre>SourceIndicatorType: type: string enum: - resourceOperation - managementOperation - unknown</pre>

Table 2 – Attributes of superclass ManagedObject_C

Attribute name	Support qualifier	Read qualifier	Write qualifier
objectClass	Mandatory	Mandatory	–
objectInstance	Mandatory	Mandatory	–
creationSource	Optional	Mandatory	–

As shown in Table 2, ManagedObject_C is made up of three attributes including objectClass, objectInstance, and creationSource. An attribute has an associated value with a specific data type. The attribute objectClass is used to identify the class type of this MO instance. The attribute objectInstance is used to uniquely identify an MO instance, and the data type is a string with the format of URI, which will be further explained in formula (1). The attribute creationSource indicates whether an MO is created automatically in a managed system, or by a managing system through a management operation, or unknown.

In network management, each MOI is uniquely identified by the object instance name. Considering the REST feature, each managed object can be regarded as a resource, and the URI as the unique identifier of the resource can naturally be the only instance of the managed object class. Resources in REST include document resources, collection resources, and task resources. The object instance is named URI string conforming to the following Backus-Naur form (BNF) paradigm specification, as shown in Table 3.

Table 3 – BNF paradigm specification for URI

<pre> URI = {URI-prefix}/{ResourcePath} URI-prefix = {irpRoot}/{irpName}/{irpVersion} ResourcePath={DocumentResourcePath} {CollectionResourcePath} {TaskResourcePath} DocumentResourcePath = ("/" {RDN})+ CollectionResourcePath=("/" {RDN})+ "/" {className} TaskResourcePath=("/" {RDN})+ ["/" {className}] "/" {actionName} RDN={className} "=" {namingAttributeValue} className = The class name of the specific MO instance. namingAttributeValue = The value of the naming attribute of the specified MO instance. </pre>	(1)
---	-----

The namingAttributeValue in the above formula should be the actual value of the naming attribute of an MOI.

A complete YAML schema definition for the generic ManagedObject_C is defined in Annex A.1.

(R) OBJECT-1. All the classes used to model resources on a managed system shall inherit (directly or indirectly) from the ManagedObject_C described above and defined in the YAML schema in clause A.1. The capabilities described above shall be supported.

8.2.2 Inheritance relationship of managed objects

Inheritance is an important concept in the object-oriented mechanism. When defining a new object class, in order to reuse the definition of the existing object class, some or all of the features defined in the existing object class can be inherited as the characteristics of the new object class. New object classes can also define additional features. In network management, all manageable managed object classes inherit directly or indirectly from the ManagedObject_C base class, and extend their own unique attribute definitions based on public attributes to more clearly express their own feature information. In YAML Schema, the inheritance of an attribute is represented by "allOf".

Taking a managed element in a telecommunication network as an example, a managed element may be composed of multiple frames, each of which includes several racks, and multiple slots are included in a rack, and circuit packs performing various functions are inserted into the slots. According to the inheritance relationship in [ITU-T M.3160], the related object classes and the inheritance relationship between each object class are shown in Figure 2.

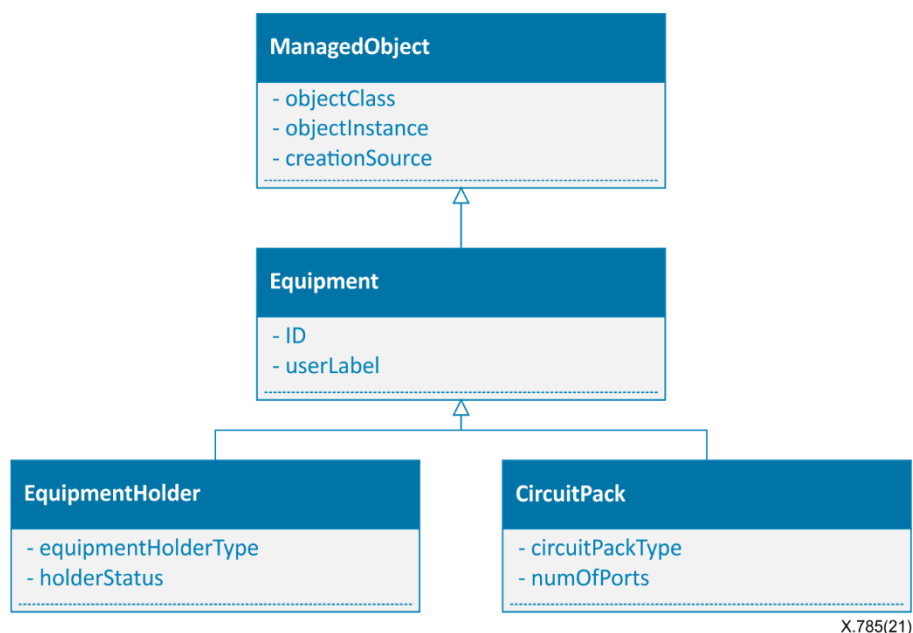


Figure 2 – An example for inheritance relationship

Table 4 – YAML schema example of Equipment and EquipmentHolder by extension

<pre>Equipment_C: allOf: - \$ref: '#/components/schemas/ManagedObject_C' - properties: ID: type: integer userLabel: type: string - required: - ID - userLabel</pre>
<pre>EquipmentHolder_C: allOf: - \$ref: '#/components/schemas/Equipment_C' - properties: equipmentHolderType: type: string holderStatus: \$ref: '#/components/schemas/HolderStatusType' - required: - equipmentHolderType - holderStatus</pre>

Some managed objects may need to support multiple inheritance, and the "allOf" syntax of the YAML schema itself supports multiple inheritance. In addition, YAML schema is only used to describe the attribute information of managed objects.

8.2.3 Common attributes and data types

The following table shows some common attributes as well as some common data types that can be shared by this framework.

Table 5 – Standard attributes and data types

Attribute name	Data type	Description
administrativeState	AdministrativeStateType	See [ITU-T M.3701] for more details
availabilityStatus	AvailabilityStatusSetType	See [ITU-T M.3701] for more details
backedUpStatus	BackedUpStatusType	See [ITU-T M.3701] for more details
controlStatus	ControlStatusSetType	See [ITU-T M.3701] for more details
creationSource (Note)	SourceIndicatorType	See [ITU-T M.3701] for more details
externalTime	ExternalTimeType	
objectClass (Note)	String	It indicates an MOC
objectInstance (Note)	URI	It indicates an MO instance
operationalState	OperationalStateType	See [ITU-T M.3701] for more details
proceduralStatus	ProceduralStatusSetType	See [ITU-T M.3701] for more details
standbyStatus	StandbyStatusType	See [ITU-T M.3701] for more details
systemLabel	String	It indicates a label for a system.
unknownStatus	UnknownStatusType	See [ITU-T M.3701] for more details
usageState	UsageState	See [ITU-T M.3701] for more details
NOTE – These attributes are inherited by all managed objects.		

The detailed YAML/JSON definitions for the above data types can be found in Annex A.1.

8.2.4 Containment relationship of managed objects

Different from the inheritance relationship, the containment relationship is more reflected in the affiliation between various network resources. As mentioned above, a switch device may contain several frames, and one frame may contain several racks. A rack may contain a number of slots, and a board that performs various functions is inserted into a slot. There may be various ports on a board. In the containment relationship, an object class (or object instance) used to contain other managed objects is called a superior, and a contained object class (or object instance) is called a subordinate. The names of the superiors and subordinates here are relative, and the subordinates of one object can be the superiors of another object. The relation type `ContainmentRelationshipType` is defined in YAML schema, which defines six attributes including the relationship name, the superior class name, the superior class multiplicity, the subordinate class name, the subordinate class multiplicity, and the naming attribute, and it is shown in Table 6.

Table 6 – The YAML schema for `ContainmentRelationshipType`

```

components:
  schemas:
    ContainmentRelationshipType:
      type: object
      properties:
        containmentRelationshipName:
          type: string
        superiorClass:
          type: string
        superiorClassMultiplicity:
          $ref: '#/components/schemas/MultiplicityType'
        subordinateClass:
          type: string
        subordinateClassMultiplicity:

```

Table 6 – The YAML schema for ContainmentRelationshipType

```
$ref: '#/components/schemas/MultiplicityType'
namingAttribute:
  type: string

DirectionType:
  type: string
  enum:
    - unidirectional
    - bidirectional

MultiplicityType:
  type: string
  enum:
    - zero_to_one
    - zero_to_n
    - one
    - one_to_n
    - n
```

Bullet (1) in Appendix II shows an example of the usage of ContainmentRelationshipType.

8.2.5 Association relationship of managed objects

In addition to the inheritance and containment relationships, there are also association relationships between managed objects, which are abstractions of network resources. An operation on one managed object may influence the attributes of another one or more managed objects. There are many types of associations, such as business relationships, control relationships, primary and secondary relationships, backup relationships, grouping relationships, peer relationships, and so on. The management system must be able to detect the existence and change of this association, and can align or coordinate the relationship through appropriate operations. Therefore, various association relationships between managed objects must be modelled.

For the definition of association relationship, the YAML schema of AssociationRelationshipType is defined. The attributes of the association relationship include association name, association direction, from association class name, from association attribute name, from association multiplicity, to association class name, to association attribute name, to association multiplicity. The types of the attributes are all string type. The YAML schema definition of the association type is shown in Table 7.

Table 7 – The YAML schema for AssociationRelationshipType

```

components:
  schemas:
    AssociationRelationshipType
      type : object
      properties:
        associationRelationshipName:
          type: string
        associationDirection:
          $ref: '#/components/schemas/DirectionType'
        fromClass:
          type: string
        fromAssociationAttribute:
          type: string
        fromMultiplicity:
          $ref: '#/components/schemas/MultiplicityType'
        toClass:
          type: string
        toAssociationAttribute:
          type: string
        toMultiplicity:
          type: '#/components/schemas/MultiplicityType'

```

Bullet (2) in Appendix II shows an example about the usage of AssociationRelationshipType.

9 Accessing methods for managed objects

There are two ways of accessing MOs in this framework. The first is to use a generic service that provides the functionality of accessing all kinds of MO instances, where a unique URI of the service is provided, as defined in clause 9.1. The other way is to access specific MOs using their own URIs, and follow the guidelines provide in clause 9.2.

9.1 Generic MO accessing methods

This clause describes the MO accessing methods of the REST-based network management framework, which shall provide a collection of methods to control network resources. These methods provide basic capabilities to manage MOs and so they are called generic accessing methods, as listed in Table 8. Figure 1 gives the accessing procedure and the framework uses HTTP protocol to exchange information of MOs. RESTful services separate an MOs' states and behaviours and expose their behaviours through a RESTful interface. As a RESTful service is also a service-oriented technology, in this framework all MOs are designed to be accessed through the REST interface, and the interface must know which MO is the actual target of an operation, and the unique identifier of the target MO should be provided in each accessing request (or through the URI). According to the above requirements, some necessary generic accessing methods are given in Table 8.

Table 8 – Generic accessing methods

Operation name	Input parameter	Output parameter
getMOAttributes	- objectClass : String - objectInstance : DN - attributeNameList : SEQUENCE OF String	- attributeNameAndValueList : SEQUENCE OF { attributeName string, attributeType string, attributeValue object } - status : ENUMERATION